Multithreaded Semantic Web Crawler

Tushar Nadkarni
Computer Engineering, Fr. Conceicao Rodrigues College of Engineering / Mumbai University, India
Email: tusharthakur1990@gmail.com

Mrs. Swati Ringe
M.E. Computer Engineering, Assistant Professor, Department of Computer Engineering
Email: swati@fragnel.ac.in

ABSTRACT

Search Engines are tremendous force multipliers for end hosts trying to discover content on the Web. As the amount of content online grows, so does dependence on web crawlers to discover relevant content. The motive is to develop an efficient Web Crawler that will give results more relevant to search keyword and faster, which will support Semantics extraction, multithreading and distributed computing.

Keywords **- About five key words in alphabetical order, separated by comma**

1) INTRODUCTION

When WWW was introduced by W3C it had limited resources for few years but in recent time there has been tremendous and rapid growth in the size of web. Google alone indexes X million pages present on web. Easy access to such huge information base and allowing public uploads is responsible for this growth. WWW, at times, is seen as huge Database but the comparison is vague as WWW is not well structured unlike database packages. There are many reasons why extracting relevant results is difficult on WWW:

A. Size of the web itself
B. Unstructured structure of Web
C. Different ways to provide the input

Web Crawler is a computer program that fetches pages from web in systematic and ordered manner. A web crawler will find the anchor links in a given page and will traverse them in an ordered fashion. Hence web crawler fetches the web page before hand and computes the relevance of the page with the help of a search keyword to eliminate pages with lesser relevance from being part of the search results. Extraction of the pages using HTTP not only depends on current processor speed but also on factors like network traffic and server load which affects the efficiency of a web crawler. Computing the relevance of a page with respect to a search keyword is also a major challenge for a web crawler, reason being, there are many ways to convey a message in a specific language and given a concept, it has many other concepts adhering to it, so directly discarding those adhered concepts while computing relevance will result in undesirable output.

The above problem is resolved with our proposed *Multithreaded Semantic Web Crawler*. Exploiting the features of java like Multithreading and RMI, to achieve high level of parallelism, helped us to improve the efficiency of a crawler. Ontology is used to discover relation among different pages that are being crawled.

2) PROPOSED SYSTEM FEATURES

2.1 URL Frontier:

It is a data structure associated with a crawler. Crawlers use URL Frontier to store the discovered URLs and process them in FIFO manner. We customize this URL Frontier to enhance the performance of the crawler. Instead of using a simple FIFO queue we assign a Priority based Queue to every thread of our crawler. Hence URL Frontier will not only store the URL but also the Priority (relevance of the URL).

getURL() -> returns the URL with Highest Priority

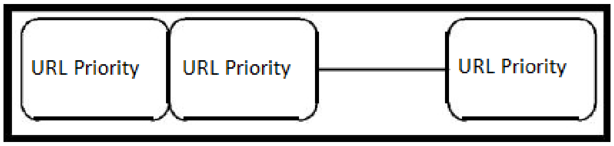

Fig 1: Structure of URL Frontier

We associate 2 Queues with each thread of the system:

1. URLDiscovered: A new URL and its relevance is added to this queue by the thread controller.

2. URLCrawled: This will store the URL that are processed by the thread(this can be a simple FIFO Queue).

2.2 Multithreading:

The idea is to achieve better parallelism by dividing the crawling process among separate independent threads. As stated earlier fetching pages from the web is time consuming and it can be carried out independently by separate threads. A Thread controller will be responsible for creating the threads and managing the URL frontiers of each thread.

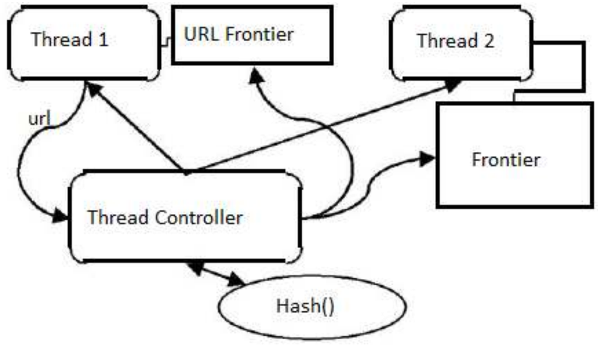

Fig 2: Multithreaded environment

Major Issues in implementing Multithreading are:

1. Duplicate Handling: We need to keep track of all the pages that are already crawled by one of the threads and avoid processing of already crawled URL. We solve the above problem with the help of thread controller. Thread controller will map every URL to unique and exactly one thread. Thread controller will see to it that the URL does not already exist in any of queue associated with that thread.

2. Combining the results: We need to sort the results of each thread as per their priority (relevance) to get the final result set. We use very efficient merge sort to sort the results of all the threads so that final result is in ascending order of relevance.

2.3 Semantic Crawler:

Relevance of a given page with respect to the search keyword can be calculated using the concept of *Ontology*. Ontology defines a common vocabulary for researchers who need to share information in a domain. It includes machine- interpretable definitions of basic concepts in the domain and relations among them.

Why would someone want to develop ontology?
Some of the reasons are [1]:

1. To share common understanding of the structure of information among people or software agents
2. To enable reuse of domain knowledge
3. To make domain assumptions explicit
4. To separate domain knowledge from the operational knowledge
5. To analyze domain knowledge

We Use Protégé [2] to develop the Ontology and Graphviz to get Graphical view of Ontology. As pointed out earlier Ontology is machine understandable description language standard which helps you to define the basic concepts in a domain and relationship between them. We use *Jena* API [3] to Query the created Ontology.

To further enhance the relevance calculation we take into account the synonyms of the search keywords. We use *WordNet and JAWS*[4] in obtain the synonyms of the word.

Working:

Step 1: Get the Hierarchy with Search keyword being the root.

Step 2: Parse the html page using Jsoup API. Get the text stored in content attribute of meta tag with name=description. Search for the occurrences of words in synonyms list for every match add 10 to relevant weight of page. Then check for occurrences of Hierarchy words in Ontology obtained in step 1, and increase weight taking in account the depth of word in hierarchy.

Step 3: Get the text stored in content attribute of body tag. Search for the occurrences of words in synonyms list for every match add 10 to relevant weight of page. Then check for occurrences of Hierarchy words in Ontology obtained in step 1, and increase weight taking in account the depth of word in hierarchy.

Step 4: Calculate the irrelevance weight of page

Step 5: Compute ratio of relevance weight to irrelevant weight

3) System Design

3.*1 Single Thread Flow:* A single thread will follow the algorithm defined below:

Step 1: Get URL with highest priority from discovered URL Frontier.

Step 2: Extract the host name 'h' from the URL, extract h/robot.txt from web initialize the vector to store disallowed URL by host h.

Step 3: Check if current URL is in Disallowed Vector. If yes go to step 1; else continue

Step 4: Fetch the page from the web.

Step 5: parse the html page to identify the anchor tags(hyperlinks in the page) for every hyperlink do:

5.1. Ask thread controller to map the URL to one of the threads.

5.2 Thread Controller assures that URL is not already processed and will map it to one of Thread if the thread is not activated it will activate it.

Step 6: Computing relevance: As described in section II.

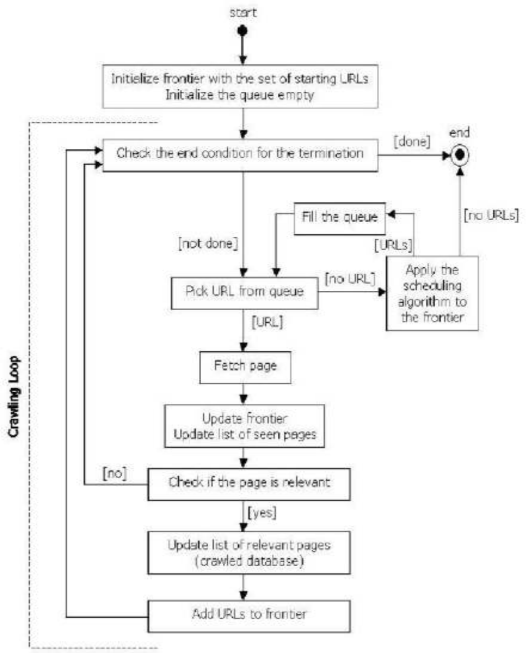
start
Initialize frontier with the set of starting URLs
Initialize the queue empty
Check the end condition for the termination
[done]
end
[not done]
Fill the queue
[no URLs]
[URLs]
Pick URL from queue
[no URL]
Apply the scheduling algorithm to the frontier
[URL]
Fetch page
Update frontier
Update list of seen pages
[no]
Check if the page is relevant
[yes]
Update list of relevant pages (crawled database)
Add URLs to frontier
Crawling Loop

Fig 3: Single thread flow

4. Performance Comparisons

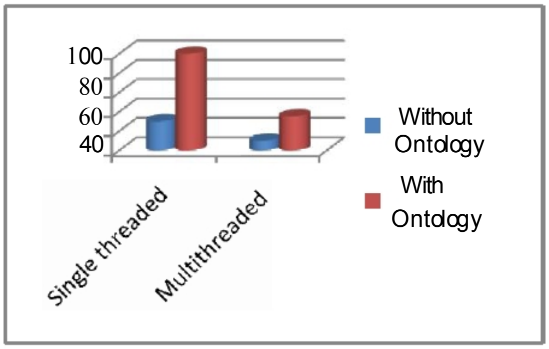

Fig 4: Comparison Chart

The Chart above shows the performance of the Web crawler while crawling for Keyword - 'authentication' in various mode of operation.

5. Conclusion

Hence, we conclude that the performance of a multithreaded and semantic web crawler was much more efficient compared to single threaded operation.

6) References

[1] Natalya F. Noy and Deborah L. McGuinness, Ontology Development 101: A Guide to Creating Your First Ontology, Stanford University, Stanford, CA, 94305

[2] Matthew Horridge, Simon Jupp, Georgina Moulton, Alan Rector, Robert Stevens, Chris Wroe, A Practical Guide To Building OWL Ontologies Using Prot´eg´e 4 and CO-ODE Tools (2007), The University Of Manchester

[3] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simpli_ed Data Processing on Large Clusters," Google Inc.

[4] Leigh Dodds, "Slug: A Semantic Web Crawler".

[5] http://jena.sourceforge.net/tutorial/index.html